MPI Tips on Cray XT5: Jaguar and Kraken

Mark Fahey

mfahey@utk.edu or faheymr@ornl.gov

NICS Scientific Computing Group Lead

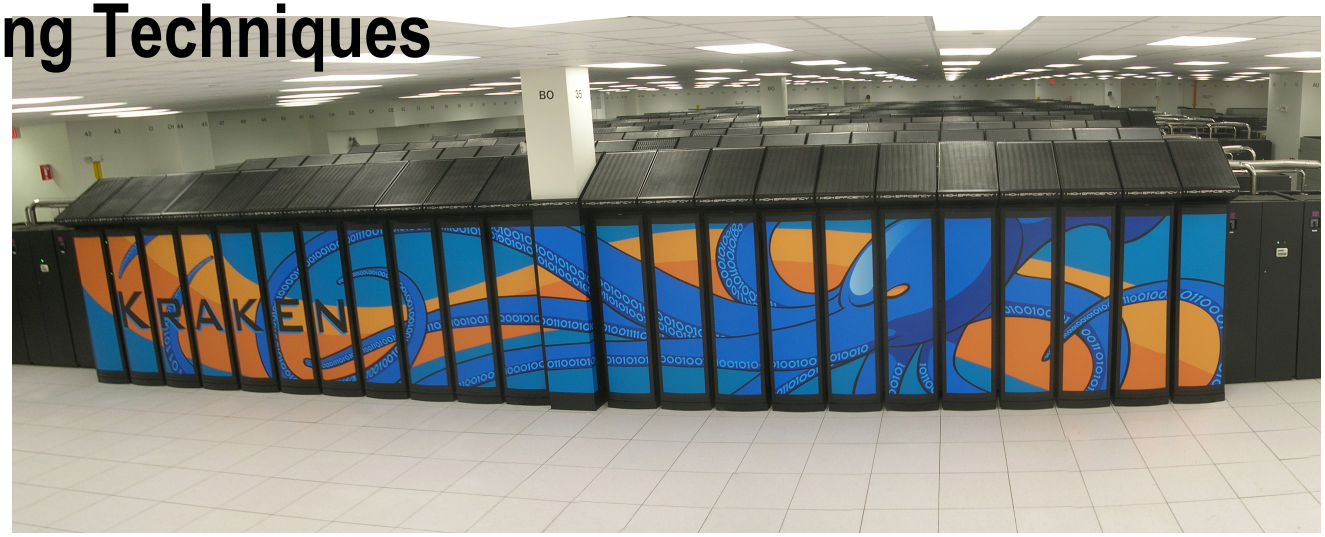
April 2009 XT Workshop

Overview

- Assuming knowledge of MPI
- Assuming you have run on an XT (3, 4, or 5)
  - Or a cluster with MPICH
- Will show ways to improve performance of MPI [on a Cray XT5]
  - Some examples from both Jaguarpf and Kraken
  - No silver bullets though
  - Nothing about MPI buffer sizes and their heuristic settings

Outline

- MPT
- Environment Variables
- Rank Placement
  - CrayPAT help
- MPI Programming Techniques
- OpenMP
- Other



MPT – Cray's MPI library

- Use latest MPT (3.1.x)
  - Significant improvements
- Many users have been/are setting env vars to set buffer sizes for 3.0 or before
- 3.1 attempts to set the right buffer sizes at launch time
  - Rather than static settings
- Suggestion: if you use env vars based on MPT 3.0 or earlier, comment them out and try out 3.1 w/o env vars
- Status:
  - Kraken: default 3.1.0 Jaguar: default 3.1.0

Environment Variables
MPI

- Next few slides will cover environment variables that are associated with MPI [performance]
- Default settings are set based on the best performance on most codes.
  - Some codes may benefit setting or adjusting environment variable settings.
- Find much of this information with “man mpi”
  - Yes, you should read the MPI man page!
- As shown on previous slide, the MPI environment changed fairly significantly, thus it is important to re-read the MPI man pages and other related documents at Cray

Environment Variables MPICH_FAST_MEMCPY

- If set, enables an optimized memcpy routine in MPI. The optimized routine is used for local memory copies in the point-to-point and collective MPI operations.
  - This can help performance of some collectives that send large (256K and greater) messages.
    - Collectives are almost always faster
    - Speedup varies by message size
    - Example: If message sizes are known to be greater than 1 megabyte, then an optimized memcpy can be used that works well for larges sizes, but may not work well for smaller sizes.
  - Default is not enabled (because there are a few cases where causes performance degradation)
  - Ex: PHASTA at 2048 processes: reduction from 262 s to 195 s

Environment Variables MPICH_COLL_SYNC

- If set, a Barrier is performed at the beginning of each specified MPI collective function. This forces all processes participating in that collective to sync up before the collective can begin.
  - To enable this feature for all MPI collectives, set the value to 1. *Default is off.*
- Can be enabled for a selected list of MPI collectives
- There are *rare* examples where this helps
  - If the code has lots of collectives and MPI profiling shows imbalance (lots of sync time), this *may* help
  - Ex: PHASTA (CFD-turbulent flows) many MPI_Allreduce calls
    - At 2048 processes : reduction from 262 sec to 218 sec.
  - Ex: But slowed down NekTarG (CFD-Blood Flow) by about 7%

Environment Variables MPICH_MPIIO_HINTS

- If set, override the default value of one or more MPI-IO hints. This also overrides any value set in the application code with calls to the MPI_Info_set routine.
- The hints are applied to the file when it is opened with an MPI_File_open() call.
- MPICH_MPIIO_HINTS_DISPLAY
  - If set, causes rank 0 in the participating communicator to display the names and values of all MPI-IO hints that are set for the file being opened with the MPI_File_open call.

Default settings:

```
PE 0:   MPIIO hints for
  c2F.TILT3d.hdf5:
    cb_buffer_size        = 16777216
    romio_cb_read         = automatic
    romio_cb_write        = automatic
    cb_nodes              = #nodes/8
    romio_no_indep_rw     = false
    ind_rd_buffer_size    = 4194304
    ind_wr_buffer_size    = 524288
    romio_ds_read         = automatic
    romio_ds_write        = automatic
    direct_io             = false
    cb_config_list        = *:1
```

Environment Variables MPICH_MPIIO_HINTS (cont.)

Examples:

- **Syntax**
  - `export MPICH_MPIIO_HINTS=data.hdf5:direct_io=true`
- **For FlashIO at 5000 processes writing out 500MB per MPI thread, the following improved performance:**

```
romio_cb_write = "ENABLE"
romio_cb_read = "ENABLE”
cb_buffer_size = 32M
```

  - **When enabled, all collective reads/writes will use collective buffering. When disabled, all collective reads/writes will be serviced with individual operations by each process. When set to automatic, ROMIO will use heuristics to determine when to enable the optimization.**
- **For S3D at 10K cores:**

`romio_ds_write = ‘disable'` **- specifies if data sieving is to be done on read.**
Data sieving is a technique for efficiently accessing noncontiguous regions of data
`romio_no_indep_rw = 'true'` **- specifies whether deferred open is used.**

  - **Romio docs say that this indicates no independent read or write operations will be performed. This can be used to limit the number of processes that open the file.**

Environment Variables MPICH_MPIIO_CB_ALIGN

- If set to 1, new algorithms that take into account physical I/O boundaries and the size of I/O requests are used to determine how to divide the I/O workload when collective buffering is enabled.
  - This can improve performance by causing the I/O requests of each collective buffering node (aggregator) to start and end on physical I/O boundaries and by preventing more than one aggregator making reference to any given stripe on a single collective I/O call.
  - If set to zero or not defined, the algorithms used prior to MPT release 3.1 are used.
  - Default: not set

Environment Variables MPICH_ENV_DISPLAY

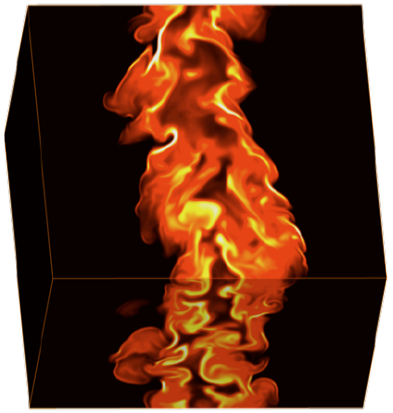
- If set, causes rank 0 to display all MPICH environment variables and their current settings at MPI initialization time.
- Default: Not enabled.
- Useful for debugging purposes.

- MPICH_VERSION_DISPLAY - displays the version of cray mpt being used

Environment Variables MPICH_SMP_OFF

- If set, disable the on-node SMP device and use the Portals device for all MPI message transfers
- Use in a rare cases where code benefits from using Portals matching instead of MPI matching.
- Default: Not enabled.
- Useful for debugging reproducibility issues.

Environment Variables
Buffer Sizes (said I wouldn't do this)



Based on experience running S3D up to 150,000 cores

- **MPICH_UNEX_BUFFER_SIZE often runs out of space**
  - **When this buffer size cannot be increased sufficiently, MPICH_MAX_SHORT_MSG_SIZE should be reduced.**
  - **Making this smaller switches the threshold for short vs long messages. Long messages are not received unless they are expected (a receive is already posted).**
  - **There is a performance penalty due to reducing the max short message size, but it will get it working.**
- **MPICH_PTL_UNEX_EVENTS and MPICH_PTL_OTHER_EVENTS have a low default value.**
  - **They are almost never adequate for large jobs. The following are good at O(10 thousand) cores.**

```
MPICH_PTL_UNEX_EVENTS=400000
MPICH_PTL_OTHER_EVENTS=100000
```

- **When an error that says 'MPI_MSGS_PER_PROC' is not sufficient is received, increase MPICH_MSGS_PER_PROC. It is an error in the error message.**
- **Buffer size variables can be set using k, M and G - Instead of having to type powers of 2 or count zeroes.**

```
% export MPICH_UNEX_BUFFER_SIZE=1G  #sets it to 1gigabyte
```

Environment Variables

MPICH_PTL_MATCH_OFF

- If set, disables registration of receive requests with portals.
  - Setting this allows MPI to perform the message matching for the portals device. It may be beneficial to set this variable when an application exhausts portals internal resources and for latency-sensitive applications.
  - Example: Used for LS-DYNA

MPICH_PTL_SEND_CREDITS

- Enables flow control to prevent the Portals event queue from being overflowed.
  - Value of '-1' should prevent queue overflow in any situation
  - Should only be used as needed, as flow control will result in less optimal performing code. If the Portals unexpected event queue can not be increased enough, then flow control may need to be enabled.

Environment Variables MPICH_PTL_MATCH_OFF

- **Case where MPICH_PTL_MATCH_OFF fixed an MPI problem**

```
[3683] : (/tmp/ulib/mpt/nightly/3.0/042108/xt/trunk/
  mpich2/src/mpid/cray/src/adi/ptldev.c:2693)
PtlMEMDPost() failed : PTL_NO_SPACE
```

- **For this, try MATCH, OTHER_EVENTS or SEND_CREDITS env var**

```
[43] MPICH PtlEQPoll error (PTL_EQ_DROPPED): An event
  was dropped on the OTHER EQ handle.  Try increasing
  the value of env var MPICH_PTL_OTHER_EVENTS (cur
  size is 2048).
aborting job:
PtlEQPoll/PtlEQGet error
```

  - **Attempts to increase OTHER_EVENTS did not help though (in this case)**

Rank Placement

- In some cases changing how the processes are laid out on the machine may affect performance, by relieving synchronization/ imbalance time.
- The default is currently SMP-style placement. This means that for or a multi-node core, sequential MPI ranks are placed on the same node.
  - In general, MPI codes perform better using SMP placement Nearest neighbor
  - Collectives have been optimized to be SMP aware
- For example, an 8-process job launched on a XT5 node with 2 quad-core processors would be placed as:

PROCESSOR	0	1
RANK	0,1,2,3	4,5,6,7

Rank Placement

- The default ordering can be changed using the following environment variable:

  MPICH_RANK_REORDER_METHOD

- These are the different values that you can set it to:

  0: Round-robin placement. Sequential MPI ranks are placed on the next node in the list.

  1: SMP-style placement. All cores from all nodes are allocated in a sequential order.

  2: Folded rank placement. Similar to default ordering except that the tasks N+1 ... 2N are mapped to slave cores of nodes N ... 1.

  3: Custom ordering. The ordering is specified in a file named MPICH_RANK_ORDER.

- When to use?
  - Point-to-point communication consumes significant fraction of program time and load imbalance detected
  - Also shown to help for collectives (alltoall) on subcommunicators (GYRO)
  - Spread out IO across nodes (POP)

Rank order and CrayPAT

- One can also use the CrayPat performance measurement tools to generate a suggested custom ordering.
  - Available if MPI functions traced (-g mpi or –O apa)
  - pat_build –O apa my_program
    - see Examples section of pat_build man page
- pat_report options:
  - mpi_sm_rank_order
    - Uses message data from tracing MPI to generate suggested MPI rank order. Requires the program to be instrumented using the pat_build -g mpi option.
  - mpi_rank_order
    - Uses time in user functions, or alternatively, any other metric specified by using the -s mro_metric options, to generate suggested MPI rank order.

Reordering Workflow

- module load xt-craypat/4.4.1
- Rebuild your code
- pat_build –O apa a.out
- Run a.out+pat
- pat_report –Ompi_sm_rank_order a.out+pat+…sdt/ > pat.report
- Creates MPICH_RANK_REORDER_METHOD.x file
- Then set env var MPICH_RANK_REORDER_METHOD=3 AND
- Link the file MPICH_RANK_ORDER.x to MPICH_RANK_ORDER
- Rerun code

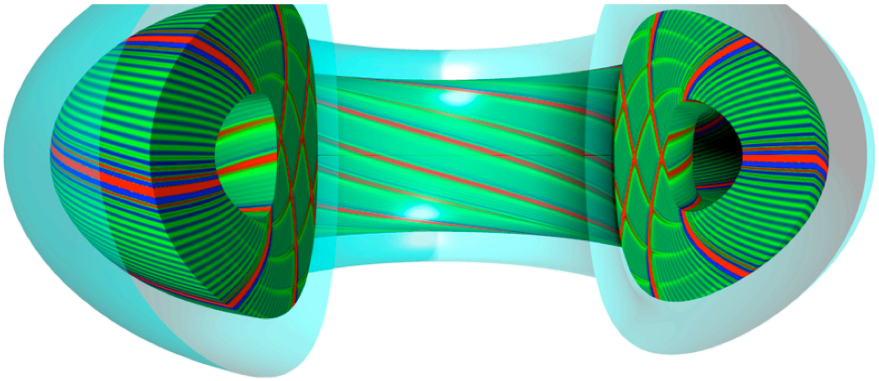
CrayPAT example

```
Table 1:  Suggested MPI Rank Order


 Eight cores per node:  USER Samp per node
 Rank          Max     Max/          Avg     Avg/  Max Node
Order  USER Samp       SMP  USER Samp       SMP  Ranks
    d       17062    97.6%       16907   100.0%  832,328,820,797,113,478,898,600
    2       17213    98.4%       16907   100.0%  53,202,309,458,565,714,821,970
    0       17282    98.8%       16907   100.0%  53,181,309,437,565,693,821,949
    1       17489   100.0%       16907   100.0%  0,1,2,3,4,5,6,7
```

- This suggests that
  1. the custom ordering “d” might be the best
  2. Folded-rank next best
  3. Round-robin 3rd best
  4. Default ordering last

Reordering example GYRO



- GYRO 8.0
  - B3-GTC problem with 1024 processes
- Run with alternate MPI orderings
  - Custom: profiled with with –O apa and used reordering file MPICH_RANK_REORDER.d

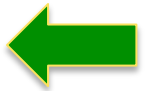
Reorder method	comm time
Default	11.26s
0 – round-robin	6.94s
2 – folded-rank	6.68s
d-custom from apa	8.03s

CrayPAT suggestion almost right!

Reordering example TGYRO

- TGYRO 1.0
  - Steady state turbulent transport code using GYRO, NEO, TGLF components
- ASTRA test case
  - Tested MPI orderings at large scale
  - Originally testing weak-scaling, but found reordering very useful

Reorder method	TGYRO wall time (min)		
	20480	40960	81920
Default	99m	104m	105m
Round-robin	66m	63m	72m



Huge win!

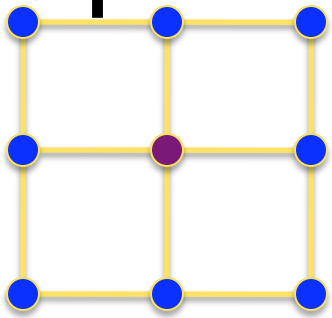
MPI Programming Techniques
Pre-posting receives

- If possible, pre-post receives before sender posts the matching send
  - Optimization technique for all MPICH installations
    - Not just an XT
- Don't go crazy pre-posting receives though. Will hit Portals internal resource limitations eventually.
- Even an IBM manual states:
  - "well-written applications try to pre-post their receives." And they also warn about posting too many.
- Code example
  - Halo update – with four buffers (n,s,e,w), post all receive requests as early as possible. Makes a big difference on CNL (not as important on Catamount).

MPI Programming Techniques
Overlapping communication with computation

- Corollary of pre-posting receives
- Use non-blocking send/recvs when it is possible to overlap communication with computation
- In some cases it may be better to replace collective operations with point to point communications to overlap communication with computation
  - Caution: Not suggesting every collective be reprogrammed by hand
  - It may be that a certain part of your algorithm has computation that could overlap the point to point communications that would not happen with a [blocking] collective

MPI Programming Techniques
Example: 9-pt stencil pseudo-code



Basic

```
9 pt computation

Update ghost cell
 boundaries
   East/West IRECV,
    ISEND, WAITALL
   North/South IRECV,
    ISEND, WAITALL
```

Maximal Irecv preposting

```
Prepost all IRECV

9 pt computation

Update ghost cell
 boundaries
   East/West ISEND,
    Wait on E/W IRECV
    only
   North/South ISEND,
    Wait on the rest
```

*Makes use of temp buffers

Example: 9-pt stencil update

```
!compute stencil
…

!update ghost cell boundaries.
!East/West
MPI_IRECV(XOUT(1,1), 1, mpi_ew_type, nbr_west,
   mpitag_wshift, COMM_OCN, request(3))
MPI_IRECV(XOUT(iphys_e+1,1), 1, mpi_ew_type,
   nbr_east, mpitag_eshift, COMM_OCN, request(4))
MPI_ISEND(XOUT(iphys_e+1-num_ghost_cells,1), 1,
   mpi_ew_type, nbr_east, mpitag_wshift, COMM_OCN,
   request(1))
MPI_ISEND(XOUT(iphys_b,1), 1, mpi_ew_type,
   nbr_west, mpitag_eshift, COMM_OCN, request(2))
MPI_WAITALL(4, request, status)

!North/South
MPI_IRECV(XOUT(1,jphys_e+1), 1, mpi_ns_type,
   nbr_north, mpitag_nshift, COMM_OCN, request(3))
MPI_IRECV(XOUT(1,1), 1, mpi_ns_type, nbr_south,
   mpitag_sshift, COMM_OCN, request(4))
MPI_ISEND(XOUT(1,jphys_b), 1, mpi_ns_type,
   nbr_south, mpitag_nshift, COMM_OCN, request(1))
MPI_ISEND(XOUT(1,jphys_e+1-num_ghost_cells), 1,
   mpi_ns_type, nbr_north, mpitag_sshift,
   COMM_OCN, request(2))
MPI_WAITALL(4, request, status)
```

```
! Prepost receive requests
MPI_IRECV(buf_west_rcv, buf_len_ew,
   MPI_DOUBLE_PRECISION, nbr_west, &
   mpitag_wshift, COMM_OCN, request(7))
MPI_IRECV(buf_east_rcv, buf_len_ew,
   MPI_DOUBLE_PRECISION, nbr_east, mpitag_eshift,
   COMM_OCN, request(8))
MPI_IRECV(XOUT(1,jphys_e+1), buf_len_ns,
   MPI_DOUBLE_PRECISION, nbr_north, mpitag_nshift,
   COMM_OCN, request(5))
MPI_IRECV(XOUT(1,1), buf_len_ns,
   MPI_DOUBLE_PRECISION, nbr_south, mpitag_sshift,
   COMM_OCN, request(6))
! compute stencil
…
! send east-west boundary info
MPI_ISEND(buf_east_snd, buf_len_ew,
   MPI_DOUBLE_PRECISION, nbr_east, mpitag_wshift,
   COMM_OCN, request(1))
MPI_ISEND(buf_west_snd, buf_len_ew,
   MPI_DOUBLE_PRECISION, nbr_west, mpitag_eshift,
   COMM_OCN, request(2))
MPI_WAITALL(2, request(7), status_wait)

! send north-south boundary info
MPI_ISEND(XOUT(1,jphys_e+1-num_ghost_cells),
   buf_len_ns, MPI_DOUBLE_PRECISION, nbr_north,
   mpitag_sshift, COMM_OCN, request(3))
MPI_ISEND(XOUT(1,jphys_b), buf_len_ns,
   MPI_DOUBLE_PRECISION, nbr_south, mpitag_nshift,
   COMM_OCN, request(4))
MPI_WAITALL(6, request, status_wait)
```

MPI Programming Techniques Aggregating data

- For very small buffers, aggregate data into fewer MPI calls (especially for collectives)
  - Ex. 1 alltoall with an array of 3 reals is clearly better than 3 alltoalls with 1 real
  - Do not aggregate too much. The MPI protocol switches from an short (eager) protocol to a long message protocol using a receiver pull method once the message is larger than the eager limit. This limit is by default 128000 bytes, but it can be changes with the MPICH_MAX_SHORT_MSG_SIZE environment variable. The optimal size for messages most of the time is less than the eager limit.
- Example – DNS
  - Turbulence code (DNS) replaced 3 AllGatherv's by one with a larger message resulting in 25% less runtime for one routine

MPI Programming Techniques
Aggregating data: Example from CFD

Original

```
    for (index = 0; index < No; index++){
       double tmp;
       tmp = 0.0;
       out_area[index] = Bndry_Area_out(A,
    labels[index]);
       gdsum(&outlet_area[index],1,&tmp);
    }
    for (index = 0; index < Ni; index++){
      double tmp;
      tmp = 0.0;
      in_area[index] = Bndry_Area_in(A,
    labels[index]);
      gdsum(&inlet_area[index],1,&tmp);
    }


void gdsum (double *x, int n, double *work)
    {
      register int i;
      MPI_Allreduce (x, work, n, MPI_DOUBLE,
    MPI_SUM, MPI_COMM_WORLD);
      /* *x = *work; */
      dcopy(n,work,1,x,1);
      return;
    }
```

Improved

```
    for (index = 0; index < No; index++){
       out_area[index] = Bndry_Area_out(A,
labels[index]);
    }

    /* Get gdsum out of for loop */
    tmp = new double[No];
    gdsum (outlet_area, No, tmp);
    delete tmp;

    for (index = 0; index < Nin; index++){
      in_area[index] = Bndry_Area_in(A,
labels[index]);
    }

    /*  Get gdsum out of for loop */
    tmp = new double[Ni];
    gdsum(inlet_area, Ni, tmp);
    delete tmp;
```

OpenMP

- When does it pay to add/use OpenMP in my MPI code?
  - Add/use OpenMP when code is network bound
  - As collective and/or point-to-point time increasingly becomes a problem, use threading to keep number of MPI processes per node to a minimum
  - Be careful adding OpenMP to memory bound codes
    - Can hurt performance
  - It is code/situation dependent!
- Rebecca Hartman-Baker talked about OpenMP Monday
  - Just reinforcing one topic here

OpenMP
aprun depth

- Must get "aprun –d" correct
  - -d (depth) Specifies the number of threads (cores) for each process. ALPS allocates the number of cores equal to depth times processes.
  - The default depth is 1. This option is used in conjunction with the OMP_NUM_THREADS environment variable.
  - Also used to get more memory per process
    - Get 1 or 2 GB limit by default (machine dependent)
  - Many have gotten this wrong, so it is important to understand how to use it properly!
    - The problem is if you don't do it right a hybrid OpenMP/MPI code can get multiple threads spawned on the same core which can be disastrous.

OpenMP aprun depth (cont.)

All on core 0
All on 1 node

One thread per core as desired!!!

```
% setenv OMP_NUM_THREADS 4
% aprun -n 4 -q ./omp1 | sort
Hello from rank 0, thread 0, on nid00291. (core affinity = 0)
Hello from rank 0, thread 1, on nid00291. (core affinity = 0)
Hello from rank 0, thread 2, on nid00291. (core affinity = 0)
Hello from rank 0, thread 3, on nid00291. (core affinity = 0)
Hello from rank 1, thread 0, on nid00291. (core affinity = 1)
Hello from rank 1, thread 1, on nid00291. (core affinity = 1)
Hello from rank 1, thread 2, on nid00291. (core affinity = 1)
Hello from rank 1, thread 3, on nid00291. (core affinity = 1)
Hello from rank 2, thread 0, on nid00291. (core affinity = 2)
Hello from rank 2, thread 1, on nid00291. (core affinity = 2)
Hello from rank 2, thread 2, on nid00291. (core affinity = 2)
Hello from rank 2, thread 3, on nid00291. (core affinity = 2)
Hello from rank 3, thread 0, on nid00291. (core affinity = 3)
Hello from rank 3, thread 1, on nid00291. (core affinity = 3)
Hello from rank 3, thread 2, on nid00291. (core affinity = 3)
Hello from rank 3, thread 3, on nid00291. (core affinity = 3)
```

```
% setenv OMP_NUM_THREADS 4
% aprun -n 4 -d 4 -q ./omp | sort
Hello from rank 0, thread 0, on nid00291. (core affinity = 0)
Hello from rank 0, thread 1, on nid00291. (core affinity = 1)
Hello from rank 0, thread 2, on nid00291. (core affinity = 2)
Hello from rank 0, thread 3, on nid00291. (core affinity = 3)
Hello from rank 1, thread 0, on nid00291. (core affinity = 4)
Hello from rank 1, thread 1, on nid00291. (core affinity = 5)
Hello from rank 1, thread 2, on nid00291. (core affinity = 6)
Hello from rank 1, thread 3, on nid00291. (core affinity = 7)
Hello from rank 2, thread 0, on nid00292. (core affinity = 0)
Hello from rank 2, thread 1, on nid00292. (core affinity = 1)
Hello from rank 2, thread 2, on nid00292. (core affinity = 2)
Hello from rank 2, thread 3, on nid00292. (core affinity = 3)
Hello from rank 3, thread 0, on nid00292. (core affinity = 4)
Hello from rank 3, thread 1, on nid00292. (core affinity = 5)
Hello from rank 3, thread 2, on nid00292. (core affinity = 6)
Hello from rank 3, thread 3, on nid00292. (core affinity = 7)
```

Other IO

- Also note that sometimes IO (especially at scale) causes scalability issues
  - See Crosby's talk on IO, Wednesday at 9:30
    - Ex. how cleaning up some writes improved weak scaling of a CFD code NektarG from 70% to 95% at 1K to 8K cores

Conclusions/Last words

- Env vars are an easy way to improve performance
  - They may not always be applicable
- Good MPI programming practices are beneficial
  - Pre-posting receives important
  - Aggregating data
- Rank reordering can significantly improve performance
- Use depth option with OpenMP or for extra memory
- Be cognizant of how IO affects your overall scalability
- Some of this may not show a benefit at <1K processes, but can reap huge wins at 10K to 100K processes
- This will become a "MPI Tips" webpage

References

- Best I have seen on Env Vars to date:
  - Geir Johansen's presentation and paper from CUG 2008
  - "Managing Cray XT MPI Runtime Environment Variables to Optimize and Scale Applications"

Extras

MPI Programming Techniques
Example: 9-pt stencil update

```
!original
do j=jphys_b,jphys_e
    do i=iphys_b,iphys_e
       XOUT(i,j) = CC(i,j)*X(i,j) + CN(i,j)*X(i,j+1) +  CN(i,j-1)*X(i,j-1) +
 &              CE(i,j)*X(i+1,j) +  CE(i-1,j)*X(i-1,j) +
 &              CNE(i,j)*X(i+1,j+1) + CNE(i,j-1)*X(i+1,j-1) +
 &              CNE(i-1,j)*X(i-1,j+1) + CNE(i-1,j-1)*X(i-1,j-1)
    end do
end do

!update ghost cell boundaries.
MPI_IRECV(XOUT(1,1), 1, mpi_ew_type, nbr_west, mpitag_wshift, COMM_OCN, request(3))
MPI_IRECV(XOUT(iphys_e+1,1), 1, mpi_ew_type, nbr_east, mpitag_eshift, COMM_OCN, request(4))

MPI_ISEND(XOUT(iphys_e+1-num_ghost_cells,1), 1, mpi_ew_type, nbr_east, mpitag_wshift,
   COMM_OCN, request(1))
MPI_ISEND(XOUT(iphys_b,1), 1, mpi_ew_type, nbr_west, mpitag_eshift, COMM_OCN, request(2))

MPI_WAITALL(4, request, status)
```

MPI Programming Techniques
Example: 9-pt stencil (cont.)

```
MPI_IRECV(XOUT(1,jphys_e+1), 1, mpi_ns_type, nbr_north, mpitag_nshift, COMM_OCN, request(3))
MPI_IRECV(XOUT(1,1), 1, mpi_ns_type, nbr_south, mpitag_sshift, COMM_OCN, request(4))


MPI_ISEND(XOUT(1,jphys_b), 1, mpi_ns_type, nbr_south, mpitag_nshift, COMM_OCN, request(1))
MPI_ISEND(XOUT(1,jphys_e+1-num_ghost_cells), 1, mpi_ns_type, nbr_north, mpitag_sshift,
   COMM_OCN, request(2))


MPI_WAITALL(4, request, status)
```

MPI Programming Techniques
Pre-posting and overlapping example

```
! Prepost receive requests
MPI_IRECV(buf_west_rcv, buf_len_ew, MPI_DOUBLE_PRECISION, nbr_west, &  mpitag_wshift,
COMM_OCN, request(7))
MPI_IRECV(buf_east_rcv, buf_len_ew, MPI_DOUBLE_PRECISION, nbr_east, mpitag_eshift,
COMM_OCN, request(8))
MPI_IRECV(XOUT(1,jphys_e+1), buf_len_ns, MPI_DOUBLE_PRECISION, nbr_north, mpitag_nshift,
COMM_OCN, request(5))
MPI_IRECV(XOUT(1,1), buf_len_ns, MPI_DOUBLE_PRECISION, nbr_south, mpitag_sshift,
COMM_OCN, request(6))

! (compute stuff)
do j=jphys_b,jphys_e
  do i=iphys_b,iphys_e
    XOUT(i,j) = CC(i,j)*X(i ,j ) + CN(i,j)*X(i ,j+1) + CN(i,j-1)*X(i ,j-1) + CE(i,j)*X(i
+1,j ) + CE(i-1,j)*X(i-1,j ) + CNE(i,j)*X(i+1,j+1) + CNE(i,j-1)*X(i+1,j-1) +
CNE(i-1,j)*X(i-1,j+1) + CNE(i-1,j-1)*X(i-1,j-1)
  end do
end do

! fill buffers and send east-west boundary info
i = 1
do n=1,num_ghost_cells
  do j=jphys_b,jphys_e
    buf_east_snd(i)=XOUT(iphys_e+n-num_ghost_cells,j)
    buf_west_snd(i)=XOUT(iphys_b+n-1,j)
    i=i+1
  end do
end do
```

MPI Programming Techniques
Pre-posting and overlapping example

```
MPI_ISEND(buf_east_snd, buf_len_ew, MPI_DOUBLE_PRECISION, nbr_east, mpitag_wshift,
COMM_OCN, request(1))
MPI_ISEND(buf_west_snd, buf_len_ew, MPI_DOUBLE_PRECISION, nbr_west, mpitag_eshift,
COMM_OCN, request(2))


! receive east-west boundary info and copy buffers into ghost cells
MPI_WAITALL(2, request(7), status_wait)

i = 1
do n=1,num_ghost_cells
  do j=jphys_b,jphys_e
    XOUT(n,j) = buf_west_rcv(i)
    XOUT(iphys_e+n,j) = buf_east_rcv(i)
    i=i+1
  end do
end do

! send north-south boundary info
MPI_ISEND(XOUT(1,jphys_e+1-num_ghost_cells), buf_len_ns, MPI_DOUBLE_PRECISION, nbr_north,
mpitag_sshift, COMM_OCN, request(3))
MPI_ISEND(XOUT(1,jphys_b), buf_len_ns, MPI_DOUBLE_PRECISION, nbr_south, mpitag_nshift,
COMM_OCN, request(4))


! receive north-south boundary info
MPI_WAITALL(6, request, status_wait)
```

MPI Programming Techniques
Example: 9-pt stencil – Yoshi optimizations

```
do j=jphys_b,jphys_e
  do i=iphys_b,iphys_e
      XOUT(i,j) = CC(i,j)*X(i,j) + CN(i,j)*X(i,j+1) +  CN(i,j-1)*X(i,j-1) + CE(i,j)*X(i+1,j) +
&         CE(i-1,j)*X(i-1,j) + CNE(i,j)*X(i+1,j+1) + CNE(i,j-1)*X(i+1,j-1) + CNE(i-1,j)*X(i-1,j+1) +
&         CNE(i-1,j-1)*X(i-1,j-1)
  end do
end do

! update ghost cell boundaries.
!fill buffers and send east-west boundary info
i = 1
do n=1,num_ghost_cells
   do j=jphys_b,jphys_e
          buf_east_snd(i)=XOUT(iphys_e+n-num_ghost_cells,j)
          buf_west_snd(i)=XOUT(iphys_b+n-1,j)
          i=i+1
   end do
end do

MPI_ISEND(buf_east_snd, buf_len_ew, MPI_DOUBLE_PRECISION, nbr_east, mpitag_wshift, COMM_OCN, request(1))
MPI_ISEND(buf_west_snd, buf_len_ew, MPI_DOUBLE_PRECISION, nbr_west, mpitag_eshift, COMM_OCN, request(2))
```

MPI Programming Techniques
Example: 9-pt stencil – Yoshi optimizations

```
!receives east-west boundary info and copy buffers into ghost cells
MPI_RECV(buf_west_rcv, buf_len_ew, MPI_DOUBLE_PRECISION, nbr_west, mpitag_wshift, COMM_OCN, status)
MPI_RECV(buf_east_rcv, buf_len_ew, MPI_DOUBLE_PRECISION, nbr_east, mpitag_eshift, COMM_OCN, status)
MPI_WAITALL(2, request, status_wait)

i = 1
do n=1,num_ghost_cells
  do j=jphys_b,jphys_e
      XOUT(n,j)         = buf_west_rcv(i)
      XOUT(iphys_e+n,j) = buf_east_rcv(i)
      i=i+1
end do; end do

!send north-south boundary info
MPI_ISEND(XOUT(1,jphys_e+1-num_ghost_cells), buf_len_ns, MPI_DOUBLE_PRECISION, nbr_north,
     &               mpitag_sshift, COMM_OCN, request(3))
MPI_ISEND(XOUT(1,jphys_b), buf_len_ns, MPI_DOUBLE_PRECISION, nbr_south, mpitag_nshift, COMM_OCN, request(4))

!receive north-south boundary info
MPI_RECV(XOUT(1,jphys_e+1), buf_len_ns, MPI_DOUBLE_PRECISION, nbr_north, mpitag_nshift, COMM_OCN, status)
MPI_RECV(XOUT(1,1), buf_len_ns, MPI_DOUBLE_PRECISION, nbr_south, mpitag_sshift, COMM_OCN, status)

MPI_WAITALL(2, request(3), status_wait)
```